Chris Piech and Mehran Sahami
CS 106A

Handout #4
April 8, 2020

Using Karel with PyCharm

Once you have downloaded a copy of PyCharm as described in the "Installing PyCharm" handout on the course website, your next task is to understand how to write and run Karel programs using PyCharm. At the end of the quarter, we will teach you how to create new PyCharm projects from scratch, but for now, providing starter code reduces the complexity of your assignments and helps you get started more easily. That way, you can ignore the mechanical details of making new projects and can focus instead on the problem-solving aspects of the assignments.

Downloading starter projects

The first step of working with any Karel assignment is to download the starter project for that assignment. If you go to the CS106A website and click on the Assignments dropdown at the top, you'll see the Karel assignment listed. Click on "1 - Karel," and on the resulting page, there will be a link to download the starter code. Once you click on this link, you'll be directed to do a short quiz on the Honor Code. After completing the quiz, you will be presented with a link that you should click on to download the starter code, which comes in a file called **Assignment1.zip**. In some cases, the browser will also automatically unzip/extract the folder when you download it if you have the appropriate software for expanding files from a ZIP archive (extraction software is usually built in to Windows 7, 8, and 10 or macOS). The unzipped contents of the ZIP file will be a directory named **Assignment1** that contains the project. **Note: If you are on a Windows computer, you may have a second Assignment1 folder nested inside the outer Assignment 1 folder. Make sure to use the innermost Assignment 1 folder.** Move that folder to some place on your computer where you can keep track of it when you want to load the project.

Importing projects into the workspace

From here, your next step is to open PyCharm, which will bring up the window shown on the left-hand side of Figure 1 (below). To select the project you want to work on, you can click on "Open" in this window. Or if you already have PyCharm open with a different project, from the "File" menu at the top of the screen, select "Open".

Navigate to the **Assignment1** folder and open it. Open the ***folder***, rather than a particular file you want to edit. If you already have another project open, PyCharm will prompt you to either open the project in your existing window or in a new window. You can select either. Once you have opened the project, you should see a view as shown in the right-hand side of Figure 1 (below).

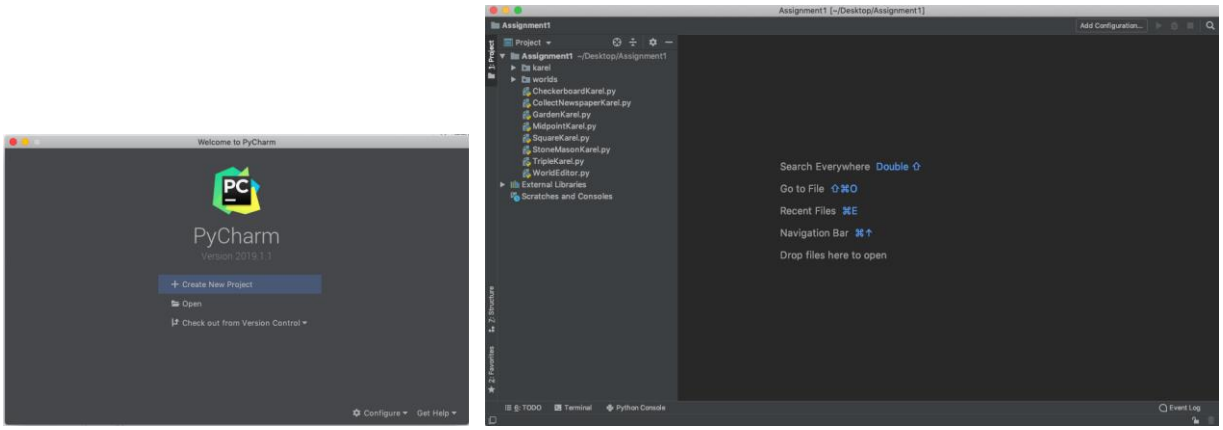


Figure 1: The first window that opens when you start PyCharm (left) should have an Open option for you to select a project. After opening a project, you should see the default PyCharm landing view (right).

First, we will focus on the contents of the left toolbar, which is illustrated in Figure 2. We can see the current project we are working on (Assignment1), as well as all of its contents, which include folders and Python files. The small triangle (which appears as a plus-sign in some versions of Windows) to the left of the folder name indicates that you can open it to reveal its contents. For the purposes of Assignment1, you will not need to inspect or change any files in the **`karel`** or **`worlds`** folders.

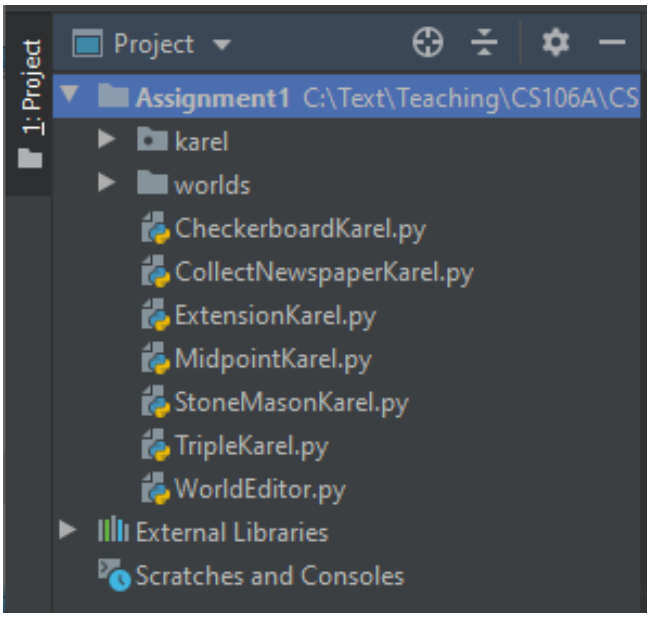


Figure 2: Left sidebar displaying **Assignment1** files

For Assignment 1, we will only focus on the files contained in the top-level of the project folder. You can open any of these files by double-clicking on its name. If you double-click on **`CollectNewspaperKarel.py`**, for example, the file will open in the editing area in the middle section of the PyCharm screen, as displayed in Figure 3.

CollectNewspaperKarel.py

```
from karel.stanfordkarel import *

"""
File: CollectNewspaperKarel.py
------------------------------
At present, the CollectNewspaperKarel file does nothing.
Your job in the assignment is to add the necessary code to
instruct Karel to walk to the door of its house, pick up the
newspaper (represented by a beeper, of course), and then return
to its initial position in the upper left corner of the house.
"""


def main():
    """
    You should write your code to make Karel do its task in
    this function. Make sure to delete the 'pass' line before
    starting to write your own code. You should also delete this
    comment and replace it with a better, more descriptive one.
    """
    pass


# There is no need to edit code beyond this point

if __name__ == "__main__":
    run_karel_program()

```

Figure 3: Text editor view of **`CollectNewspaperKarel.py`**

Note that sometimes the comments at the top of the file may not display initially and may need to be "expanded" by clicking the small '**+**' sign (to the left of the comment header lines and to the right of the line numbers).

The file we include in the starter code contains not the finished product but only the header line for the project. The actual program must still be written. If you look at the assignment handout, you'll see that the problem is to get Karel to collect the "newspaper" from outside the door of its "house" as shown in Figure 4.

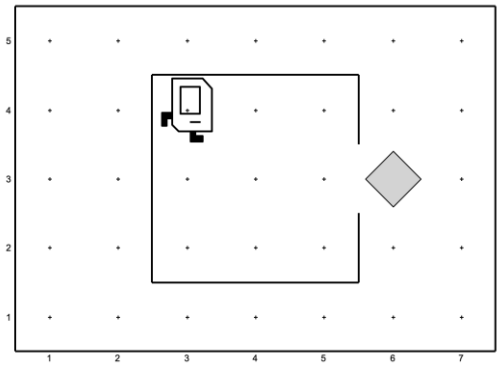
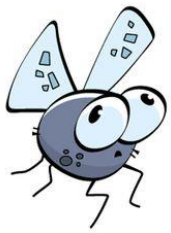


Figure 4: Karel's starting state for **`CollectNewspaperKarel.py`**

Suppose that you just start typing away and fill in the **`main`** method with the steps below:

```
def main():
    move()
    turn_karel_right()
    move()
    turn_left()
    move()
    pick_beeper()
```



This program isn't going to do exactly what you want (hence the bug picture off to the side)! But it is still interesting to see what happens. PyCharm reads through your program file as you type commands and then tells you about any errors it finds. Errors will be underlined in red, and you can get more information about a given error by hovering over it (shown in Figure 5).

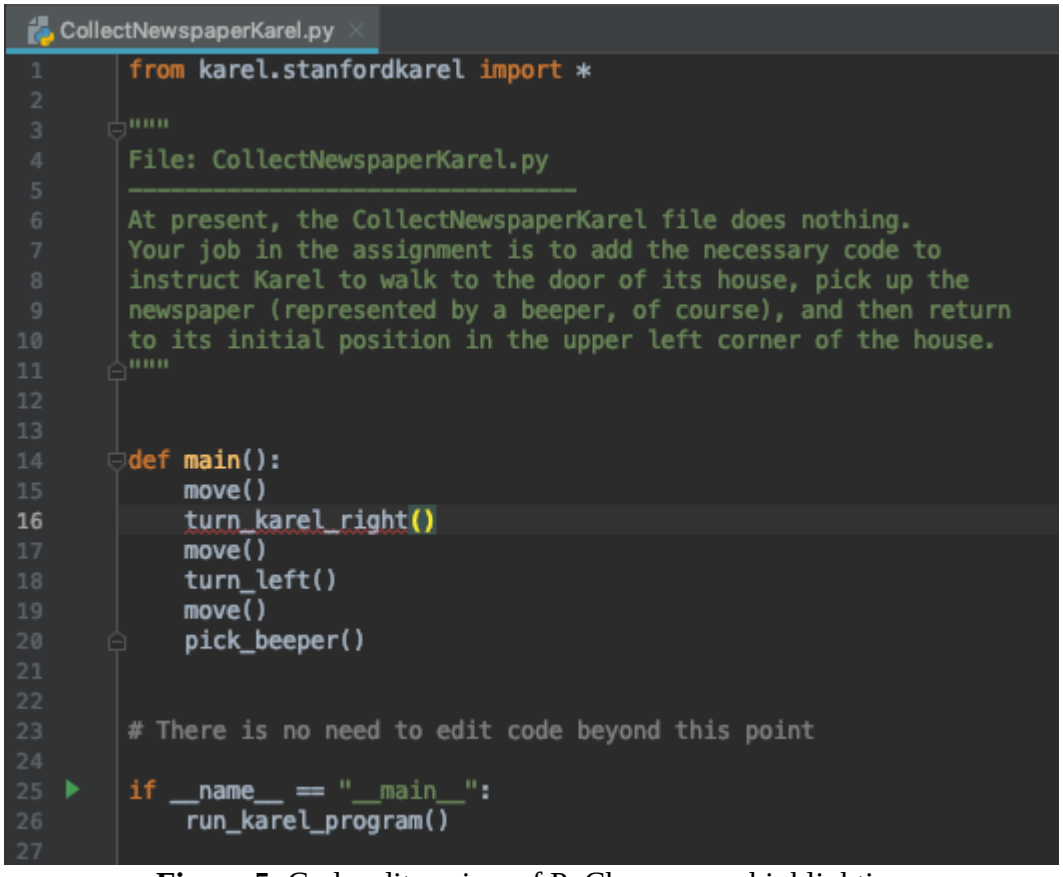


Figure 5: Code editor view of PyCharm error highlighting

In this case, an error of the type "Unresolved reference" tells us that we tried to give Karel the command **`turn_karel_right()`** but never defined what that command means! To fix this error, we should add a function definition in the file that defines what it means to

`turn_karel_right()`. Once we fix this error, we see that PyCharm no longer underlines any of the lines in our program in red!

Running a Karel program in PyCharm

To run a program in PyCharm, click the "Terminal" option at the bottom of the screen, which should open up a portion of PyCharm as seen in Figure 6.

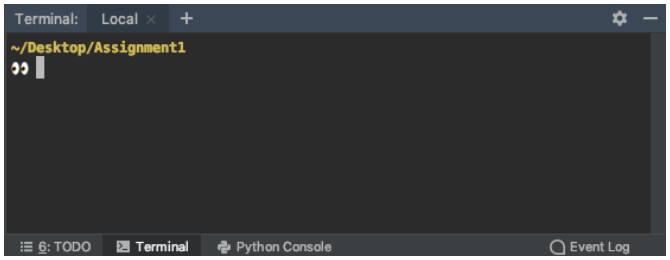


Figure 6: The PyCharm terminal that is used to run Python programs

Mac Users: To run any Karel program, all you have to do is type the following command into the Terminal and hit "Enter":

`python3 InsertNameHereKarel.py`

PC Users: To run any Karel program, all you have to do is type the following command into the Terminal and hit "Enter":

`py InsertNameHereKarel.py`

As a side note, in many handouts we might use the Mac convention of running programs by using the command **`python3`** <***program name***>. If you are a PC user, you should instead use the command **`py`** <***program name***>. Just by convention on the PC, you run the Python interpreter using the command **`py`** (as opposed to **`python3`** on the Mac).

For example, if we were working on a Mac and wanted to run **`CollectNewspaperKarel.py`**, we would type in the command shown in Figure 7 and then hit "Enter".

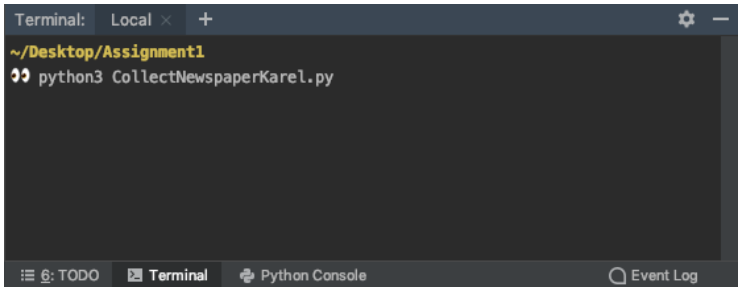


Figure 7: The PyCharm terminal with the correct command to run **`CollectNewspaperKarel`**

PyCharm will take a couple of seconds to load your program and then will display a window as seen in Figure 8.

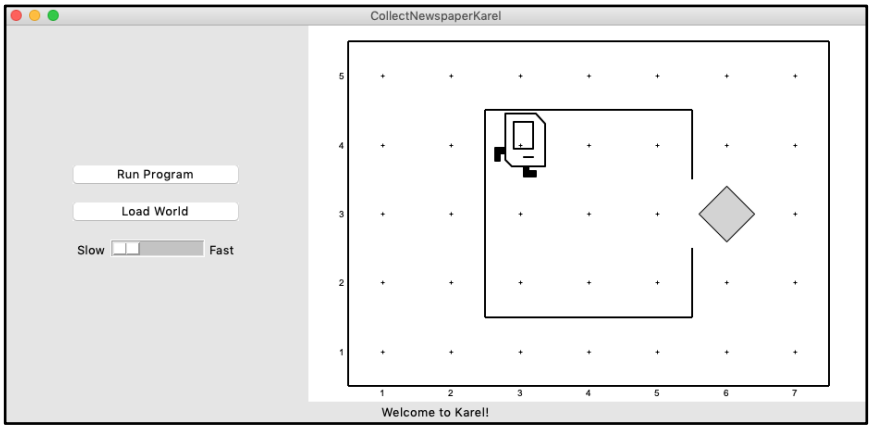


Figure 8: The Karel display that appears when running **`CollectNewspaperKarel.py`**, including Karel's world, a **Run Program** button, and a **Load World** button

If you then press the **Run Program** button, Karel will go through the steps in the **`main()`** function that you wrote.

To change the speed at which Karel runs through its commands, you can use the slider underneath the two buttons. Moving the slider to the right increases the speed at which Karel runs, while moving the slider to the left causes Karel to move slower. Setting the slider to the far-right will cause Karel to almost instantly finish its task, with all intermediate action happening too quickly for the human eye to distinguish.

In this case, however, all is not well. Karel begins to move across and down the window as if trying to exit from the house, but ends up one step short of the beeper. When Karel then executes the **`pick_beeper()`** command at the end of the **`main()`** method, there is no beeper to collect. As a result, Karel stops, a popup window appears, red text is displayed at the bottom of the window, and an error is displayed in PyCharm's bottom panel, as shown in Figures 9 and 10.

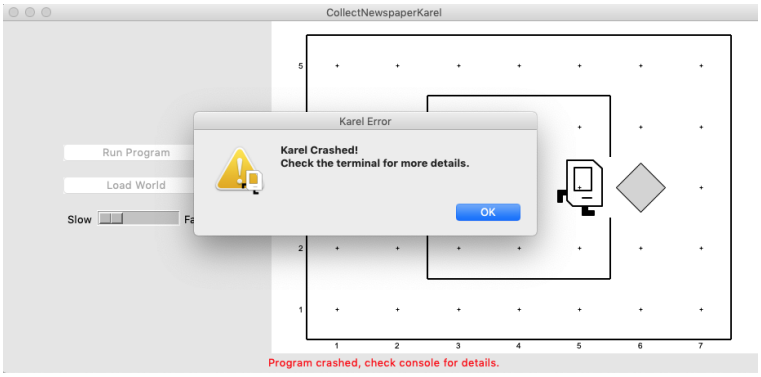


Figure 9: All is not well when you get a notification that Karel has crashed

Figure 10: The bottom panel of PyCharm displays an error message that occurs when something has gone wrong in Karel's world

This is an example of a **logic bug**, in which you have correctly followed the syntactic rules of the language but nonetheless have written a program that does not correctly solve the problem. The error message doesn't tell you how to fix the issue, but if you look at what it says carefully, you'll see that it does tell you why Karel stopped (Karel tried to pick up a beeper on corner (5,3) when "no beepers on corner") and what line of your program Karel was executing when the error occurred ("line 24 in main" in **`CollectNewspaperKarel.py`**).

Running into bugs like these is a very common thing for all programmers, and though the text looks scary, you are equipped with all the tools you need to think through the issue and to update your code accordingly. More information on how to work through these kinds of logic errors can be found in the Debugging section of this handout. To make Karel run again, after you've made changes to your code, you can close the original window and run the same command as before to re-run the program.

Running Karel in different worlds

In order to successfully complete Assignment 1, the last piece of the puzzle that we haven't worked through yet is how to run your Karel program in different worlds. Say we are working on **`TripleKarel`** and we think that we have gotten our code to work in the default world, which looks like what is shown in Figure 11.

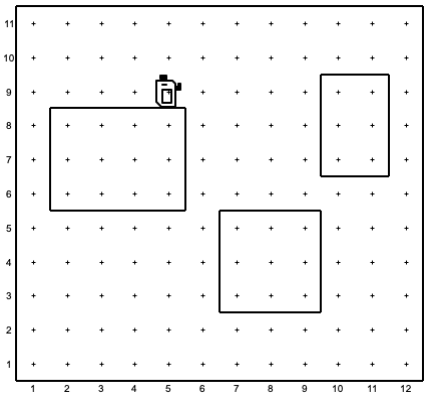


Figure 11: TripleKarel's default world

We want to make sure that Karel works in different worlds so that the code we wrote is bug-free and generalizes to new worlds. To do so, click on the **Load World** button, which brings up a file browser view that lists all of the provided worlds that come packaged with Assignment 1, as shown in Figure 12.

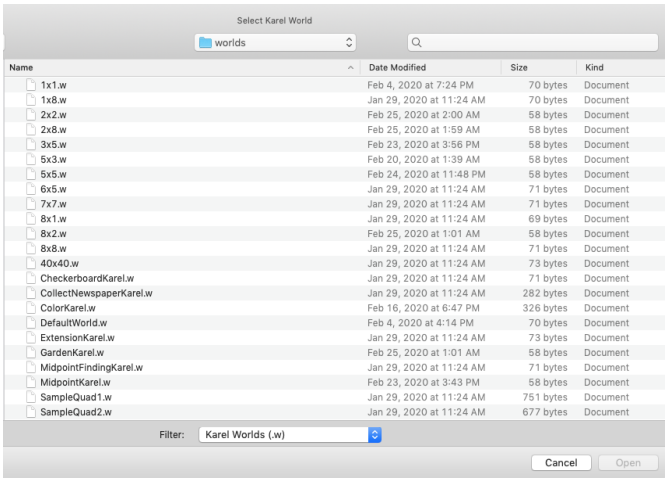


Figure 12: File browser window showing all the world files in the Assignment 1 **`worlds`** folder.

Select and open the world in which you want to run Karel. The new world will now appear on the right hand side of the Karel window. You can then test Karel in this new world by clicking on the **Run Program** button.

Note: If you see the following error appear when loading a new world, you can safely ignore it.
objc[98046]: Class FIFinderSyncExtensionHost is implemented in both /System/Library/PrivateFrameworks/FinderKit.framework/Versions/A/FinderKit (0x7fff883583c8) and /System/Library/PrivateFrameworks/FileProvider.framework/OverrideBundles/FinderSyncCollaborationFileProviderOverride.bundle/Contents/MacOS/FinderSyncCollaborationFileProviderOverride (0x115d89f50). One of the two will be used. Which one is undefined.

Debugging

> *"As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs."*
>
> — Maurice Wilkes, 1979

More often than not, the programs that you write will not work exactly as you planned and will instead act in some mysterious way. In all likelihood, the program is doing precisely what you told it to. The problem is that what you told it to do wasn't correct. Programs that fail to give correct results because of some logical failure on the part of the programmer are said to have **bugs;** the process of getting rid of those bugs is called **debugging**.

Debugging is a skill that comes only with practice. Even so, it is never too early to learn the most important rule about debugging:

> *In trying to find a program bug, it is far more important to understand what your program is doing than to understand what it isn't doing.*

Most people who come upon a problem in their code go back to the original problem and try to figure out why their program isn't doing what they wanted. Such an approach can be helpful in some cases, but it is more likely that this kind of thinking will make you blind to the real problem. If you make an unwarranted assumption the first time around, you may make it again and be left in the position that you can't for the life of you see why your program isn't doing the right thing.

When you reach this point, it often helps to try a different approach. Your program is doing *something*. Forget entirely for the moment what it was supposed to be doing, and figure out exactly what is happening. Figuring out what a wayward program is doing tends to be an easier task, mostly because you have the computer right there in front of you. PyCharm has many tools that help you monitor the execution of your program and figure out what is going on. You'll have a chance to learn more about these tools in the coming weeks.